Closest Strings, Primer Design, and Motif Search

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Jens Gramm Falk Hüffner Rolf Niedermeier

Wilhelm Schickard-Institut für Informatik
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany

{gramm,hueffner,niedermr}@informatik.uni-tuebingen.de

1 Introduction

CLOSEST STRING
Given: k strings from Σ^L and a positive integer d.
Question: Find a "closest string" s such that none of the given strings has *Hamming* distance greater than d to s.

Example (d_H denotes the Hamming distance):

$s_1 =$ TAGTGTATT
$s_2 =$ TGTTGTATT
$s_3 =$ TAGTGTGCT
$s_4 =$ TGGTGTGTT

$s =$ TGGTGTGTT — "closest string" with $d_H(s, s_i) \leq 2$ for $i = 1, 2, 3, 4$

- CLOSEST STRING is NP-complete.

Here, we exploit two new techniques for solving CLOSEST STRING (partially presented in [4]) and provide experimental evidence to show that they are practically useful for primer design and motif search (the experiments reported here are made on a SUN workstation with Ultrasparc IIe processor with 500 MHz and 512 MB main memory).

2 Exact Algorithms for CLOSEST STRING

In [4], we presented new algorithmic strategies for solving CLOSEST STRING and showed an application in primer design. We summarize and update some results:

Search tree algorithm

- Running time $O(kL + kd \cdot d^d)$, i.e., the exponential growth of the running time bounded by a function in d only. This is of importance for biological applications where d tends to be a small number.
- E.g., randomly generated instances with $L = 50$, $d = 20$, $k = 50$ can be solved in 100 sec and instances with $d < 10$ (and reasonable values of L and k) in less than 1 sec.

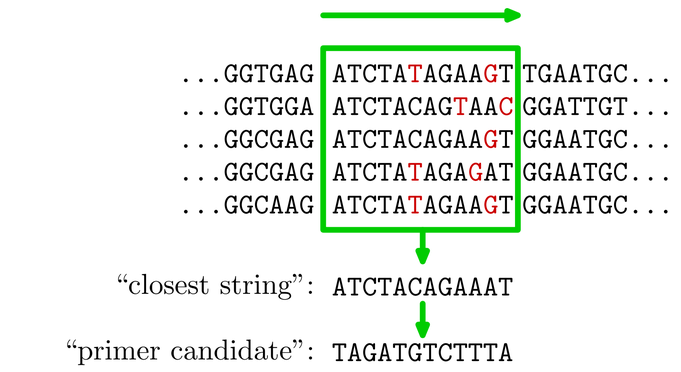
Integer Linear Programming

- Ben-Dor *et al.* [1]: ILP formulation using $L \cdot |\Sigma|$ many variables.
- In [4]: ILP formulation in which the variable number depends only on the number of input strings.
- Both formulations can be solved efficiently (on randomly generated data for $L \leq 100, k \leq 20$ in less than 1 sec) using heuristic branch and bound techniques for ILPs; e.g., we used the GNU GLPK library.
- Experiments indicate that the second ILP formulation is preferable for instances with a small number of long input strings.

3 Application in Primer Design.

Algorithms for CLOSEST STRING can be used to design primers that bind to all sequences from a given set of homologous sequences.
Strategy to design primers of length L that bind to each sequence with at most d mismatches:

1. Compute an alignment of these sequences.
2. "Slide" a length L window over all the aligned strings, solving a CLOSEST STRING instance for every window position. Using the search tree algorithm, (2) can be done in time $O(kL + (n - L)kd \cdot d^d)$ (i.e., linear time for constant d) where n is the length of the alignment.



The search tree algorithm is advantageous:

- In this application, parameter d usually has a small value.
- It is easy to incorporate constraints desirable for primers, e.g., disallowing mismatches in initial positions of the primer or searching solutions favorable in terms of melting temperature.
- An extension of the algorithm can deal with the situation when we are given an additional set of sequences to which the primer should *not* bind.

4 Application in Motif Search

In motif search, given a set of k strings and parameters L and d, the objective is to find an (L, d)-*motif*, which is a string of length L such that each of the given strings has a length L substring with distance at most d to the motif.

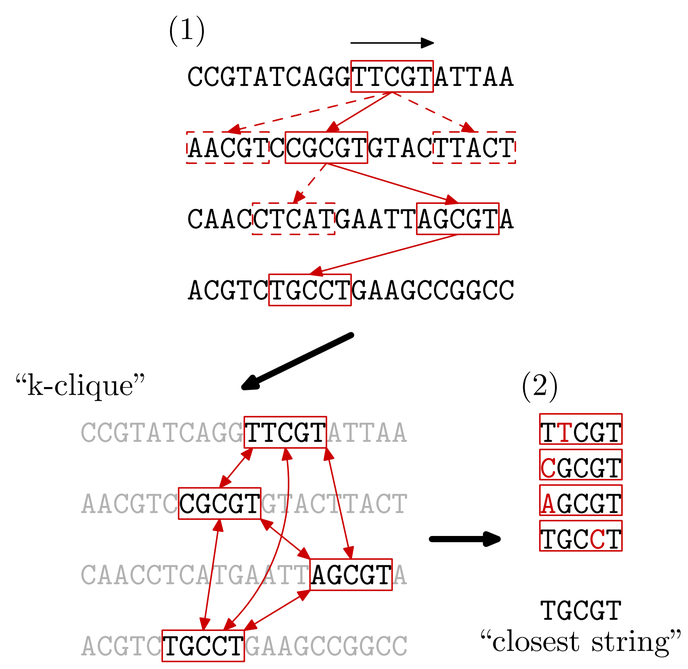
Enumerative Algorithm

This algorithm uses CLOSEST STRING as a subproblem. Similar to [5], in a first phase our algorithm enumerates candidates, where, in our case, the "filtering process" is more elaborate since, in a second phase, we check the outcome of the first phase:

1. Identify the "k-cliques" of substrings, taking one substring from each given string, with pairwise distance at most $2d$ each time.
2. For each such candidate set, use a CLOSEST STRING algorithm to test whether it gives rise to a motif.

Step 1 is a recursive algorithm that enumerates all possible sets of substrings: For each length L substring s_1 of the first input string, it considers each length L substring s_2 of the second input string with distance at most d to s_1. For each such pair s_1, s_2 it considers each length L substring s_3 of the third string with distance at most d to s_1 and s_2, and continues recursively.

Example with $k = 4$ given strings, motif length $L = 5$, and $d = 1$ allowed mismatches (red arrows denote a Hamming distance of at most $2d = 2$):



Refinement. E.g., if two substrings in the set have distance exactly $2d$, we can restrict the set of possible closest strings: each of its characters must match the corresponding character in one of the two substrings.

CLOSEST STRING **algorithm.** With parameter values in practical settings, both described CLOSEST STRING approaches are efficient; for our experiments, we used the ILP.

Spurious Motifs. Our algorithm finds all positions in the strings that meet the specified requirements and will report no positions that do not; this is its advantage in contrast to algorithms in [5] and [3]. However, it clearly cannot exclude to find spurious motifs, e.g., random motifs in artificially generated data that have not been implanted. Note that not all combinations of L and d values are reasonable; if d is chosen too large compared to L, we can expect spurious random motifs [3], e.g., with 20 random sequences of length 600, one can expect at least one $(15, 5)$-motif.

Performance. Compared to the algorithm in [2], ours is less sensible to growing values of L and d; however, it is more sensible to growing number and length of the input sequences.

Experiments — Artificial Data

The artificially generated data we used are random strings containing one implanted motif. The enumerative algorithm can find an implanted (L, d)-motif in 20 strings of length 600 for different L and d values that were posed as "challenge problems" in [5] or [3]:

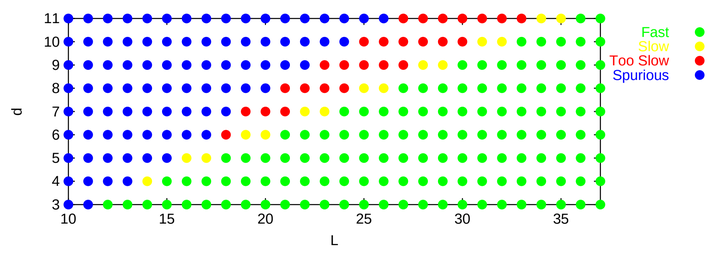
L	d	running time	L	d	running time	L	d	running time
13	3	6 sec	15	4	40 sec	17	5	5 min
14	4	7 min	16	5	1 h 57 min	19	6	59 min

Hard instances. The (18,6) problem, harder for the algorithm in [3], turns out to be particularly hard for us also.

Longer motifs. We solve, e.g., the (29,9) problem in 2 minutes.

Longer input sequences. We solve, e.g., the (15,4) problem for strings of length 1000 in 4 minutes.

Performance on 20 strings of length 600 with different parameter values for L and d (green = processing time less than 1 min; yellow = processing time up to a few hours; red = instances unsolvable due to high processing time; blue = instances unsolvable due to spurious motifs):



Experiments — Real Data

We also tried data from the ACUTS gene database, which contains sets of homologous noncoding sequences with 5–10 sequences of length 500–1000 per set. Our algorithm performs quite good on these data; for example, we can answer the question "Given a value d, what is the longest value for L for which we can find an (L, d) motif?" within seconds for every sample.

This research was supported by the DFG, research project OPAL (optimal solutions for hard problems in computational biology), NI 369/2-1.

References

[1] A. Ben-Dor, G. Lancia, J. Perone, R. Ravi. Banishing Bias from Consensus Sequences. In *Proc. of the 8th CPM*, number 1264 in LNCS, pages 247–261, 1997. Springer.

[2] M. Blanchette. Phylogenetic Footprinting. In *Proc. of the 5th RECOMB*, pages 49–58, 2001. ACM Press.

[3] J. Buhler and M. Tompa. Finding motifs using random projections. In *Proc. of the 5th RECOMB*, pages 67–74, 2001. ACM Press.

[4] J. Gramm, R. Niedermeier, and P. Rossmanith. Exact solutions for Closest String and related problems. In *Proc. of the 12th ISAAC*, number 2223 in LNCS, pages 441–453, 2001. Springer.

[5] P. A. Pevzner and S.-H. Sze. Combinatorial approaches to finding subtle signals in DNA sequences. In *Proc. of 8th ISMB*, pages 269–278, 2000. AAAI Press.